API 文档缺陷自动检测和修复方法 *

王长志 [1]，周　宇 [1, 2†]，闫　鑫 [1]

(1. 南京航空航天大学 计算机科学与技术学院，南京 210016; 2. 南京大学 软件新技术国家重点实验室，南京 210093)

摘　要：应用程序编程接口(application programming interface, API)在软件开发中起着非常关键的作用，而 API 文档的质量会严重影响开发人员对 API 的使用。为了完善 API 文档，提出了基于程序静态分析和自然语言处理的自动检测和修复 API 文档缺陷的方法，该方法能自动检测和修复 API 文档缺陷。实验中，缺陷检测结果的准确率和召回率分别达到 74.6%和 81.4%，能够较为准确地检测到 java API 的文档缺陷。在进一步的实验中，还对 API 文档的修复功能进行了评估，结果表明生成的文档正确且简洁，可以有效地修复 API 文档缺陷。

关键词：Java API 文档；程序异常；修复建议

中图分类号：TP311.53　　doi: 10.3969/j.issn.1001-3695.2018.07.0368

Novel approach to automatically detect and repair defective API documentation

Wang Changzhi[1], Zhou Yu[1, 2†], Yan Xin[1]

(1. *College of Computer Science & Technology*, *Nanjing University of Aeronautics & Astronautics*, *Nanjing* 210016, *China*; 2. *State Key Laboratory for Novel Software Technology*, *Nanjing University*, *Nanjing* 210093, *China*)

Abstract: Application Programming Interface (API) play an important role with respect to software development. Particularly, the quality of API documentation tends to have a severe impact on the use of APIs for developers. In this paper, for improving API documentation, a novel approach is proposed to automatically detect and repair defective API documentation by adopting techniques of program static analysis and natural language processing. Across empirical studies, the proposed approach gives a precision of 74.6% and a recall of 81.4%, which demonstrates that the approach can accurately detect the defective documentation in java APIs. For further experiments, the function of repair recommendation for API documentation is evaluated. The results indicate that the generated recommendations are correct and succinct, and can repair API documentation effectively.

Key words: Java API documentation; Exceptions; Repair recommendation

0　引言

随着计算机应用的不断深化，软件的需求和规模不断增加。如何快速高效的完成软件开发成为软件行业的追求目标。有效的复用是提高软件开发效率、降低开发成本的重要方式，其中，应用程序编程接口（API）复用是最有效的手段之一[1]。开发人员无须了解 API 的底层实现或理解其内部工作机制的细节，只需正确利用开放的函数接口，即可实现相应的功能，从而提高了软件的开发效率。

快速正确的理解 API 是使用 API 的前提，如果理解 API 花费的时间过长甚至超过了重新编写类似功能代码所需的时间，则失去了使用 API 的意义[2]。API 文档在帮助开发人员正确使用 API 中起着至关重要的作用[3]。API 文档通过说明 API 的使用情境，如输入信息描述、版本信息、参数约束条件等，使得开发人员遵循这些规则而避免在使用 API 时出现错误[4,5]。然而，由于程序员的疏漏和软件维护升级等原因的存在，导致 API 文档存在缺陷的现象时常发生[6~8]。这些缺陷文档的存在会使得开发人员在使用 API 时陷入困境，阻碍其对 API 的理解并影响软件的性能[9,10]。例如：在 com.sun.javafx.fxml.BeanAdapter 类中的 get(Object target, Class<?> sourceType, String key)方法中，对于参数 key 的描述仅为“The property name.”但是如果给参数 key 传入一个空值 null，则会抛出一个 NullPointerException 的异常。经过分析发现，在 get()方法调用到它类中的 getStaticGetterMethod(Class<?> sourceType, String key, Class<?> targetType)方法时，有以下约束条件：如果参数 key 为 null，则抛出异常。而 get()方法会直接把参数 key 的值传入 getStaticGetterMethod()方法，从而引起了该异常。显然，在 get()方法的文档中应该存在关于该异常的描述，以避免开发人员触发该异常，导致程序出错。因此，完善的 API 文档对于开发人员而言是非常重要的。

收稿日期：2018-07-17；**修回日期：**2018-09-11　**基金项目：**国家重点研发计划资助项目（2018YFB1003902）；江苏省自然科学基金资助项目（BK20151476）

作者简介：王长志（1991-），男，山东滨州人，硕士研究生，主要研究方向为软件智能演化分析；周宇（1981-），男（通信作者），江苏徐州人，副教授，博士，主要研究方向为软件演化分析、形式化验证技术（zhouyu@nuaa.edu.cn）；闫鑫（1994-），男，河南周口人，硕士研究生，主要研究方向为软件演化.

当前，关于 API 文档缺陷检测的研究较少，其中文献[11，12]将 NLP 运用到 java 文档中。文献[13]提出了一种结合自然语言处理和代码分析技术来检测 API 文档错误的方法。文献[14]就文档中对 API 参数约束的描述进行了相关研究，基于文献[15]对文档中的约束类型进行的调查，将参数约束类型分为四类，分别为空值禁止，空值允许，类型限制和取值限制。文献[16]提出了一种利用 Stack Overflow 来提高 API 文档质量的方法，通过收集零散的信息来为开发人员提供更全面的支持。同样，在[17]中介绍了一种将来自在线网站的源代码实例连接到 API 文档的方法。本文针对的研究问题虽然与他们不同，但借鉴了他们的分析技术。

本文是对已有工作进行的扩展[18]，扩展内容体现在以下几个方面：a)实验方法针对的约束类型，由空值允许、空值禁止、取值限制，扩展为空值允许、空值禁止、取值限制、类型限制四种，使本方法更具有普适性;b)通过对 API 文档进行自然语言处理，理解其语义信息，定义并总结了 64 条启发式语义规则，从文档中提取相对应的约束信息。相比于之前方法，大大增加了文档约束信息的准确率和覆盖率;c）通过使用约束求解器对代码和文档的约束条件进行检测，使得检测结果更加准确；d)根据检测到的文档缺陷，本文新增加了缺陷修复方法。

1　方法概述

本节首先通过一个实际例子来更为直观的说明本文的目标与基本方法，之后给出基于程序静态分析的自动检测和修复 API 文档缺陷方法的总体框架。

1.1　示例说明

图 1 中是 com.sun.javafx.event.EventHandler- Manager 类中的方法。其中在 validateEventType()方法中，当“evertType == null”时，有异常抛出。显然，因为 addEventFilter()调用了 validateEventType()方法，在 addEventFilter()方法中，如果 evertType 赋值为 null 时，该异常也会抛出。因此，为防止开发人员触发该异常，API 文档中有关于该异常的描述：“@throws NullPointerException if the event type or filter is null”。

```
/**
 * Registers an event filter in {@code EventHandlerManager}.
 *
 * @param <T> the specific event class of the filter
 * @param eventType the type of the events to receive by the filter
 * @param eventFilter the filter to register
 * @throws NullPointerException if the event type or filter is null
 */
public final <T extends Event> void addEventFilter(
        final EventType<T> eventType,
        final EventHandler<? super T> eventFilter) {
    validateEventType(eventType);
    validateEventFilter(eventFilter);

    final CompositeEventHandler<T> compositeEventHandler =
            createGetCompositeEventHandler(eventType);

    compositeEventHandler.addEventFilter(eventFilter);
}
```

图 1　从 JDK 中选取的示例

本文中的研究方法即检测代码中抛异常的约束条件在文档中是否存在正确描述，如果不存在描述，则文档存在缺陷，并对文档进行修复。为解决这一问题，本文使用程序静态分析和基于模板的自然语言处理技术，具体过程分为以下几个步骤：首先，利用程序静态分析，从代码中提取抛异常时的约束条件，即“evertType == null”。其次，对 API 文档进行处理，提取与异常相关的约束信息，在本例中，从 API 文档里可以提取到有关 evertType 为 null 的信息。将两者提取的信息进行对比，判断是否一致。显然在本例中，两者约束一致，文档描述正确。假设当该文档中不存在这个描述时，则文档是有缺陷的，需要对其进行补全修复。为此，本文定义了相关的语义规则，利用从代码中提取到的约束条件，完成文档修复工作。根据规则，该约束条件生成的文档为:“@throws NullPointerException if evertType is null”。

1.2　方法框架

本文提出了一种文档缺陷自动检测和修复技术。图 2 给出了该方法的整体流程图。该方法主要由四个部分构成：

a)代码约束条件提取。该部分利用程序静态分析技术，获取方法之间的调用关系，结合调用关系，提取程序抛出异常时的约束条件。

b)文档约束条件提取。根据总结的启发式语义规则，提取文档中关于异常信息描述的约束条件。

c)代码与文档约束条件对比。将代码和文档中的约束条件转换为一阶逻辑表达式(first order logic, FOL），利用 Z3 求解器进行对比，获取逻辑验证结果，若两者不一致，则认为文档有缺陷，需要进行补全。

d)生成文档缺陷报告及修复建议：本文根据 FOL 的格式，设定了 FOL 生成自然语言的语义规则。根据代码中提取的约束条件 FOL 和语义规则，将代码约束条件转换为自然语言推荐给用户，使得用户可以对文档进行补全。

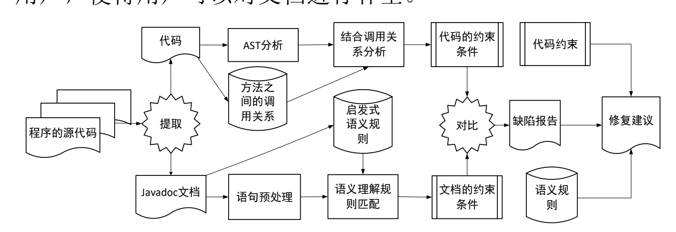


图 2　文档缺陷检测修复技术的整体流程

在现有工作中，本方法主要检测空值禁止、空值允许、取值限制和类型限制四种约束类型是否有文档缺陷[13]，以下是关于四种类型的介绍：

a)空值禁止。空值 null 不能作为参数传递给该方法。当参数传入空值时，会有异常抛出，如 NullPointerException。

b)空值允许。与空值禁止相反，空值 null 可以作为参数传递给该方法，此时没有异常抛出。因为参数为 null 是一种比较特殊的情况，因此需要在文档中予以描述。

c)取值限制。方法参数必须符合某一取值范围，如果超出这一范围，则会有异常抛出。

d)类型限制。类似于取值限制，方法参数必须属于某些类型，如果不属于这些类型，则会抛出异常。由于面向对象程序语言具有继承的特性，因此在 API 文档中也必须对此类作出准确描述。

2　方法实现

2.1　约束条件提取

对于 API 来说，异常抛出的约束条件不仅存在于其本身方法体中，也存在于它的调用方法之中，因此代码约束条件提取分为以下三步：a)方法调用关系分析，分析方法之间的调用关系，建立方法间调用关系集合;b)方法约束条件提取，构建方法体的抽象语法树，提取方法体中的约束信息;c)考虑调用关系是否存在，构建完整的约束条件信息。

2.1.1 调用关系分析

因为本文主要关注方法参数的约束条件是否在文档中予以描述，因此在分析调用关系时，仅关注直接传参类型的。例如在 javax.swing.JTabbedPane 类的 addTab(String title, Component component)方法中，Component 类型的参数最终传递给了 java.awt.Container 类中 checkNotAWindow(Component comp)方法从而引发了异常。这种即为存在参数传递的调用关系。

首先，解析 API 源码，提取方法 m 的抽象语法树，再通过 eclipse 的 call hierarchy 模块进行调用关系分析。具体分为以下几步：获取方法的 m 的参数列表 Pm，如果方法 m 调用了方法 c，获取方法 c 的参数列表 Pc。如果满足 Pm 与 Pc 的交集不为空，则方法 c 与方法 m 存在直接传参的调用关系，将方法 c 放入方法 m 的调用关系集合之中。

2.1.2 约束条件提取

对于 API 的每个 public 方法 m，通过遍历抽象语法树，获得所有的 throw 语句，收集有关的异常信息，回溯查找触发该异常的约束条件，将每个方法 m 的异常信息存储为一个元祖的集合 ExcepInfom (m;P;t;c)，其中 m 是当前方法的名称，P 是方法的参数列表，t 是异常的类型，c 是异常的触发条件。提取异常约束条件信息的伪代码如下所示：

代码中约束条件的提取算法：

```
Data: stmList:AST statement block of a method m, and dep: integer
Result: infoList:list of exception information,which records the folw-exception tuples,i.e,(m;P;t;c)
infoList ← ∅
    if dep >= 0 then
foreach stm ∈ stmList do
 /* If stm throws an exception, records all information in a tuple and add to the list */
if isThrowable(stm) then
infoList ← infoList ∪ {f(m; P; t; c) j P :
parameter, t : exceptiontype, c : condition}
 /* Recursively invoke itself, in case of composite statement */
else if isComposite(stm) then
List subList ← (Block)stm:getBody();
infoList ←
  infoList ∪ expExtractor(subList; dep);
 /* If the statement contains a method call of n, check the invoked method recursively */
else if isMethod(stm) ^ (stm's args ∈ m's list) then
 /* n is the callee of m in stm */
mList ← n:getBody();
infoList ←infoList ∪ expExtractor(mList; dep - 1);
```

2.1.3 约束条件信息整合

根据第二步中获得的每个方法 m 的 ExcepInfom 列表。首先删除与参数无关的异常约束信息，即对于每个 ExcepInfo(m;P;t;c)，如果异常触发条件 c 没有 P 中的参数出现，则此异常信息与参数无关，这种情况不在本文讨论范围之内，将此条 ExcepInfo 删除。其次，根据第一步获得的调用关系集合，对于集合中的每个方法，逐层递归获取其异常信息。即对于方法 m 和方法 n，m 调用 n，如果(n;P;t;c)中约束条件 c 与方法 m 的参数 p 有关，则方法 m 也能触发方法 n 的这个异常，(n;P;t;c)要被归并到 ExcepInfom 列表中。这样，便获得了方法 m 完整的异常约束条件信息。特别的，在递归调用时，为防止调用关系分析形成闭环，需要设置递归深度阈值，本文在实验中设置的阈值为 4。

2.2　文档约束条件提取

API 文档对约束信息描述具有自然语言特性，基于观察发现，对于相同的约束类型，API 文档的文本描述通常具有类似的语法结构。由于这种共性的存在，使得可以通过基于自然语言处理的思想和方法，对文档进行处理，提取其约束条件。其中，本文利用 Stanford Parse 对语句进行处理。Stanford Parse 是斯坦福大学开发的一款开源自然语言处理框架。该框架提供了词性标注、句法解析和词项依存关系分析等功能。除了能对英语进行处理，还能对中文和德文等语言进行解析，功能强大。

在 API 文档中，javadoc 的每一个@代表着一个标签条目，不同的标签类型其描述的内容也不相同。如：@version 标签主要描述版本说明信息，@since 指定程序代码最早使用的版本。由于本文关注点在文档中对于方法参数的约束条件描述，因此主要对@exception、@throws 和@param 这三个标签中的内容进行分析。

本方法主要包括以下两个步骤：

a)预处理阶段。删除句子中一些标记与符号。根据句子的语法树结构，将含有多个约束条件的复合语句拆分为只含有一个约束条件的原子语句。

b)语义理解。根据依存语法和启发式语义规则，对原子语句进行理解。

2.2.1 文档预处理

API 文档虽然具有自然语言的特性，但其与纯自然语言叙述有所不同，它经常会与代码的一些标识符混在一起。为将文档处理为纯自然语言，需要对文档进行预处理工作，比如标签的标题@exception、@throws 和@param（为啥这三个要被删除），还有一些嵌入式的标记（如<code>）等，都需要被删除。预处

理时，为了不丢失有效信息，它们的标签类型和后面的参数都将会被记录下来进行保存。

在获得自然语句之后，由于一个语句中可能含有多个约束条件，且每个约束所约束的行为也可能不同，会对语句的理解产生偏差，因此需要对语句进行拆分，使其变为只含有一个约束条件的原子语句。本文利用 Stanford Parse 对语句进行处理，获得语句的语法树，通过分析语法树，对语句进行拆分，得到每个约束条件的原子语句。

2.2.2 语义理解

文档预处理后，通过对获得的 API 文档的原子语句进行语义分析，提取其中的约束条件。基本思想是根据句子的依存语法和人工定义的启发式语义规则，对句子进行解析。

利用程序去正确理解自然语言是一件非常困难的事情，但对于 API 文档，由于其专业性的存在，使得很多文档具有相同的句式[19][20]。例如：javax.swing.BorderFactory 类中的 createStrokeBorder(BasicStroke stroke,Paint paint) 方法对应的 API 文档中提到，“if the specified {@code stroke} is {@code null}”，则会抛出异常。在 javax.swing.event. SwingPropertyChangeSupport 类中的 firePropertyChange(final PropertyChangeEvent evt)方法，也有类似的描述，“if {@code evt} is {@code null}”，则抛出异常。上述两个例子都是 API 文档中提到的参数不能为空的描述，其句式都为“[parameter] be/equals null”，依存语法为“mark(null-4, if-1),nsubj(null-4, parameter-2),cop(null-4, is-3),root(ROOT-0, null-4)”。利用依存语法就可以解析到“parameter == null”的约束条件。因此，可以根据句式，利用依存语法，从类似的描述中提取约束条件信息。

基于此，本文定义了识别特定结构语句的启发式语义规则，利用这些规则对 API 文档进行解析，获取文档中的约束条件。通过对 JDK 中部分 API 文档进行归纳总结，得到 4 类共 64 条语义解析规则，如表 1 所示。

表 1　启发式语义规则

规则类型	描述	数量
空值禁止	输入参数不能为空	20
空值允许	传入参数可以为空，具有特殊意义	11
取值限制	参数取值必须在一定范围内	10
类型限制	参数必须属于某些类型	23
总计		64

2.3　约束条件对比

在获得代码和文档的约束条件信息之后，将其转换为 FOL 公式表示。在处理过程中，定义一组转换规则，例如：“parameter=null”的 FOL 为：NullConstraint(parameter;NEG)；NullConstraint 代表空值约束，parameter 代表参数，NEG 代表否定（即参数为空时有异常）。

获得代码和文档的 FOL 后，利用可满足性模型理论，对两者进行验证，检测两者逻辑是否一致，从而判断文档是否有缺陷。这个判断基于的假设是：默认代码中的约束描述是正确的，即当代码与文档中的约束描述不一致时，则判定文档是有缺陷的。形式上，当两者关于参数 x 的 FOL 满足如下公式时，则认为文档是正确的：

$$\Phi_{api} \Leftrightarrow \Phi_{doc}$$

其中：Φ_{api} 代表代码中的约束条件，Φ_{doc} 代表文档的约束条件。显然，检测上述公式等于检测：$(\Phi_{api} \vee \neg\Phi_{doc}) \wedge (\neg\Phi_{api} \vee \Phi_{doc})$ 是否可被满足。根据此公式，利用 SMT(Satisfiability Modulo Theories)工具，将 FOL 有序加入，获得逻辑验证结果，检测文档是否有缺陷，并得到 API 文档的缺陷报告。

2.4　缺陷修复

由于 API 文档没有对代码中的约束条件信息进行准确描述，所以针对该缺陷将代码中的约束条件信息依据规则生成自然语言补全到 API 文档中。如图 2 所示，基于代码的 FOL 和缺陷报告生成相关修复建议。本文在归纳启发式语义规则时，已经总结了 API 文档对于每种约束的描述。在生成文档时，从中选取最为简洁的语言描述作为模板，将 FOL 转换为自然语言。共有 4 类 11 条模板，如表 2 所示。

表 2　缺陷修复模板

规则类型	Tags	模板
		If [param] be null
空值禁止	@throws	If [param1] or [param2] be null
		If [param1], ... , or [paramN] be null
空值允许	@param	[param] could be null
取值限制	@throws	If [param] be type of [SpecType]
		If [param] be not type of [SpecType]
		If [param] {relation} [value]
		If [param] {relation} [value1], ... , [valueN]
类型限制	@throws	If [param1] or [param2] {relation} [value]
		If [param] {relation} [value1] and {relation} [value2]
		If [param] {relation} [value1] or {relation} [value2]

生成的文档与 javadoc 相同，会为其添加一个标签，出于简洁性考虑，对于报错的统一使用@throws，对于空值允许的则使用@param。例如，在 com.sun.glass.ui.mac.MacFileNSURL 类的 createFromDocumentScopedBookmark(byte[] data, MacFileNSURL baseDocument) 方法中，提取的 FOL 为 NullConstraint(data, NEG)，生成的修复建议为“@throws NullPointerException　If data is null.”。最终，缺陷修复生成的文档与原有 API 文档格式一致，具有良好的可读性。

3　实验分析

由于 JDK 中的 API 文档比较完备，为了验证所提出的缺陷检测和修复方法的有效性及准确性，实验一将 JDK 部分项目的源代码为实验对象，对本方法进行验证。同时，为了进一步证明方法对其他 java API 的适用性，实验二采用 Google 公司的 guava 项目作为实验对象。

本实验的实验平台为 Intel i7-4790 3.6 GHz 处理器和 32.0

GB RAM 的台式机电脑，搭载 Windows 7 64-bit 操作系统，使用 Java 版本为 1.8.0_25，使用 Eclipse Luna-SR2 作为代码实现的开发 IDE。

3.1 实验准备

本文以准确率和召回率来衡量方法的检测结果。以人工标记为准确结果，相关计算公式如下：

$$\text{precision} = \frac{TP}{TP+FP}, \text{recall} = \frac{TP}{TP+FN}$$

$$\text{F-measure} = 2\times\frac{\text{precision}\times\text{recall}}{\text{precision}+\text{recall}}$$

其中 TP 表示方法判断文档缺陷存在的情况是真是存在的，FP 表示文档缺陷存在的情况属于误判，TN 表示方法对文档无缺陷的判断是正确的， FN 表示方法的漏判断。Precision 为准确率，表示报出文档缺陷的情况中，报告准确所占的比例。Recall 为召回率，表示源代码里所有存在的 API 文档缺陷中，被实验方法检测出的比例。F-measure 是 precision 和 recall 的调和平均数，是一种对准确率和召回率进行综合考量的指标。

3.2 实验 1

对 com.sun、javafx 和 javax.swing 三个包中的源代码作为测试对象进行了实验。经过统计，这三个包中，代码总行数约为 134 万，共有 4 万多种方法。在这些文档中，@param、@throw 和@exception 三种标签的总数为 21462 条。具体统计情况如表 3 所示。

表 3 实验 1 数据总览

源代码包	Com.sun	Javafx	Javax.swing	总计
源代码行数	344.3k	625.4k	372.8k	1342.5k
方法数	10125	19807	12601	42533
@param	2773	7832	8531	19136
@throws	271	1040	448	1759
@exception	17	17	533	567

对数据进行统计时，首先通过人工标记的方法获取一个标准数据集，文档是否有缺陷以人工判断为准。共有五名计算机专业的研究生对代码文档进行评估分析，判断文档是否有缺陷。实验结果如表 4 所示。

表 4 实验 1 检测结果

约束类型	空值禁止	空值允许	取值限制	类型限制	总计
TP	179	457	227	42	905
FP	67	51	183	7	308
FN	5	27	80	95	207
准确率(%)	72.8	90.0	55.4	85.7	74.6
召回率(%)	97.3	94.4	73.9	30.7	81.4
F-m(%)	83.3	92.1	63.3	45.2	77.8

如表 4 所示，对于不同的约束类型，本文的检测方法都能取得较好的效果，其中空值允许的准确率能够达到 90%。但是，取值限制的准确率较低，仅有 55.4%。经过对实验结果的分析发现，许多不清晰的描述降低了取值限制的准确率。例如在 javax.swing.AbstractButton 类的 checkHorizontalKey(int key, String exception)方法中，若 key 不为"LEFT、CENTER、RIGHT、LEADING 或 TRAILING"时，则会有 IllegalArgumentException 异常抛出。但相应的 API 文档的描述为"@exception IllegalArgumentException if key is not one of the legal values listed above"。由于提取不到有效信息，在方法中会判定为不一致，但人工会判定为一致。从而降低了准确率。但这种文档虽可以算作有约束描述，却描述模糊，对正确理解 API 起不到任何作用。在实验一中，缺陷检测准确率较高，达到 74.6%。

3.3 实验 2

为验证本方法的普适性，在实验 2 中选取了 Google 公司的 guava 项目作为测试对象。Guava 是该公司的一组 Java 核心库，其中，guava.com 项目代码行数约为 16 万，共有方法四千多个。@param、@throw 和@exception 三种标签的总数为 2276 条。

采用与实验一同样的评判方式，实验结果如表 5 所示。

表 5 实验 2 检测结果

约束类型	空值禁止	空值允许	取值限制	类型限制	总计
TP	545	8	89	2	644
FP	57	2	116	1	176
FN	4	4	24	1	33
准确率(%)	90.5	80.0	43.4	66.7	78.5
召回率(%)	99.3	66.7	78.8	66.7	95.1
F-m(%)	94.7	72.3	56.0	66.7	86.0

在本实验中共报出文档缺陷 820 条，其中有 644 条是准确的，176 条误报。实验结果中出现了与实验一相同的问题，即取值限制的准确率相对较低。导致这个现象的原因有很多，其中有实验一提到的原因。也有如 com.google.common.base.Preconditions 类 checkPositionIndexes(int start, int end, int size) 方法的文档中有这样的描述"@throws IndexOutOfBoundsException if either index is negative or is greater than {@code size},or if {@code end} is less than {@code start}"从人为的角度"either index"就可以看做是方法的这说明参数，但是在方法中的文档提取中，由于提取不到参数名，而认为文档没有描述。这说明本方法中使用的文档提取规则具有一定的局限性，但实验的总准确率达到 78.5%，充分证明本实验方法对 java API 的适用性，特别是对于空值约束的类型，本方法能取得较高的检测准确率。

3.4 实验 3

为了验证 API 文档缺陷修复的质量，利用以上两个实验得到的缺陷检测结果，对于有缺陷的文档生成对应的修复建议。本实验从中随机抽取 400 条修复建议作为样本，并提供相应的源代码和相关约束条件。选取五名计算机专业的研究生，对每条修复建议从准确性、内容充足性和简洁性等四个方面进行打分[21]，评估修复建议的质量，分数为 5 到 1 分，5 分为最好，3 分为中立，1 分为最差，其中每条数据评估三次。打分标准如表 6 所示。

表 6　修复文档质量评分标准

	问题	分数范围
Q1	修复建议是否准确的表示了代码约束条件？	5-1
Q2	修复建议是有利于帮助使用此 API？	5-1
Q3	修复建议是否含有与代码约束条件无关的信息？	5-1
Q4	修复建议是否简洁易懂？	5-1

评估结果如表 7 所示.

表 7　实验 3 评分结果：

结果	Q1	Q2	Q3	Q4
5	898	558	903	733
4	153	183	158	261
3	44	195	56	105
2	31	188	30	30
1	74	76	53	71
平均分	4.48	3.82	4.53	4.31

在所有样本中，评估结果为 3 分以上的修复建议占比为 88.5%，结果证明本文中的修复方法对于修复现有的文档缺陷起着积极作用，而且，除问题 2 外，平均分都在 4 分以上。问题 2 分数较低的原因在于所抽取的样本中本身就有缺陷检测属于误判的数据，对于此类数据，其 API 本身的文档可能就有较为准确的描述，因此问题 2 得分较低。由评估结果说明，本实验方法生成的修复建议质量较高，简洁明了，充分地表述了代码中的约束条件，可以有效的修复 API 文档的缺陷。

4　结束语

本文首先以 JDK 的子项目作为实验对象进行实验。通过实验 1 和 2 验证了文档缺陷检测和修复方法的合理性和准确性，检测方法准确率高，修复方法则能生成合理的缺陷修复建议，对文档进行良好的补充。在实验 2 中，验证了本方法对于 Java 语言的其他开源项目同样适用。实验结果标明对于不同约束条件类型、不同的项目，本文所提出检测和修复方法具有普遍适用性，并取得了良好的实验结果。

今后还将继续扩展归纳实验中的启发式语义规则，进一步提高方法的准确性，并将开发实现该方法的界面图形工具，如基于 Eclipse 等集成开发环境采用插件式开发[22]，提供更为用户友好的支持。同时，针对使用启发式语义规则的局限性，探究使用机器学习的方式处理 API 文档，以便在文档中提取到更多的有效信息，并在此基础上，探究本方法对于其他编程语言的适用性。

参考文献：

[1] Iyer B, Subramaniam M. The strategic value of APIs [EB/OL]. (2015-01-07) https://hbr. org/2015/01/the-strategic-value-of-apis.

[2] Myers B A. Human-centered methods for improving API usability [C]// Proc of International Workshop on API Usage and Evolution. IEEE Press, 2017: 2.

[3] Inzunza S, Juárez-Ramírez R, Jiménez S. API documentation [C]// Proc of World Conference on Information Systems and Technologies. ChamL: Springer, 2018: 229-239.

[4] Earle R H, Rosso M A, Alexander K E. User preferences of software documentation genres [C]// Proc of International Conference on the Design of Communication. New York: ACM Press, 2015: 1-10.

[5] Linares-Vásquez M, Bavota G, Penta M D, *et al.* How do API changes trigger stack overflow discussions?a study on the Android SDK [C]// Proc of the 22nd International Conference on Program Comprehension. New York: ACM Press, 2014: 83-94.

[6] Chatterjee P, Nishi M A, Damevski K, *et al.* What information about code snippets is available in different software-related documents?An exploratory study [C]// Proc of IEEE, International Conference on Software Analysis, Evolution and Reengineering. Piscataway, NJ: IEEE Press, 2017: 382-386.

[7] Zhou Yu, Gu Ruihang, Cheny T, *et al.* Analyzing APIs documentation and code to detect directive defects [C]// Proc of International Conference on Software Engineering. Piscataway, NJ: IEEE Press, 2017: 27-37.

[8] Uddin G, Robillard M P. How API documentation fails [J]. IEEE Software, 2015, 32 (4): 68-75.

[9] Arnaoudova V, Penta M D, Antoniol G. Linguistic antipatterns: what they are and how developers perceive them [J]. Empirical Software Engineering, 2015, 21 (1): 1-55.

[10] Endrikat S, Hanenberg S, Robbes R, *et al.* How do API documentation and static typing affect API usability? [C]// Proc of the 36th International Conference on Software Engineering. New York: ACM Press, 2014: 632-642.

[11] Moreno L, Marcus A. Automatic software summarization: the state of the art [C]// Proc of International Conference on Software Engineering Companion. Piscataway, NJ: IEEE Press, 2017: 511-512.

[12] Panichella S, Panichella A, Beller M, *et al.* The impact of test case summaries on bug fixing performance: an empirical investigation [C]// Proc of IEEE//ACM International Conference on Software Engineering. Piscataway, NJ: IEEE, 2017: 547-558.

[13] Zhong Hao, Su Zhendong. Detecting API documentation errors [J]. ACM SIGPLAN Notices, 2013, 48 (10): 803-816.

[14] Saied M A, Sahraoui H, Dufour B. An observational study on API usage constraints and their documentation [C]// Prpc of IEEE International Conference on Software Analysis, Evolution and Reengineering. Piscataway, NJ: IEEE, 2015: 33-42.

[15] Petrosyan G, Robillard M P, De Mori R. Discovering information explaining API types using text classification [C]// Proc of IEEE//ACM International Conference on Software Engineering. Piscataway, NJ: IEEE, 2015: 869-879.

[16] Treude C, Robillard M P. Augmenting API documentation with insights from stack overflow [C]// Proc of IEEE//ACM International Conference on Software Engineering. Piscataway, NJ: IEEE, 2017: 392-403.

[17] Subramanian S, Inozemtseva L, Holmes R. Live API documentation [C]//

Proc of the 36th International Conference on Software Engineering. New York: ACM, 2014: 643-652.

[18] 古睿航, 周宇. 一种 Java API 文档对异常描述不一致的自动检测方法[J]. 计算机应用研究, 2017, 34 (7): 2032-2037. (Gu Ruihang, Zhou Yu, An automatic approach for detecting inconsistent description of exceptions in Java API documentation [J]. Application Research of Computers, 2017, 34 (7): 2032-2037.)

[19] Sorbo A D, Panichella S, Visaggio C A, *et al.* Development emails content analyzer: Intention mining in developer discussions [C]// Proc of the 30th IEEE//ACM International Conference on Automated Software Engineering. Piscataway, NJ: IEEE, 2015: 12–23.

[20] Di Sorbo A, Panichella S, Visaggio C A, *et al.* DECA: development emails content analyzer [C]// Proc of the 38th International Conference on Software Engineering Companion. New York: ACM Press, 2016: 641-644.

[21] Mcburney P W. Automatic documentation generation via source code summarization [C]// Proc of IEEE//ACM International Conference on Software Engineering. Piscataway, NJ: IEEE, 2015: 903-906.

[22] 马晓星, 曹春, 余萍, 等. 基于图文法的动态软件体系结构支撑环境[J]. 软件学报, 2008, 19 (8): 1881-1892. (Ma Xiaoxing, Cao Chun, Yu Ping, *et al.* A Supporting environment based on graph grammar for dynamic software architectures [J]. Journal of Software, 2008, 19 (8): 1881-1892.)